Reitinetsintä ja etsintäavaruudet tietokonepeleissä

Timo Maaranen

Helsinki 01.03.2015
Pro gradu -tutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta – Fakultet –Faculty	Laitos – Institution – Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä – Författare – Author

Timo Maaranen

Työn nimi – Arbetets titel – Title

Reitinetsintä ja etsintäavaruudet tietokonepeleissä

Oppiaine – Läroämne – Subject

Tietojenkäsittelytiede

Työn laji – Arbetets art – Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages
Pro gradu -tutkielma	01.03.2015	35

Tiivistelmä – Referat – Abstract

Opinnäytetyö esittelee moderneissa peleissä usein käytettyjä reitinetsintämenetelmiä sekä etsintäavaruuksia, joissa menetelmät toimivat. Se perehdyttää reitinetsinnän periaatteisiin peruasioista lähtien ja sopii siten esimerkiksi johdannoksi aiheeseen. Näkökulmana on käytännönläheinen ja siinä pyritään ottamaan huomioon pelilaitteiden rajoitukset, etsintämenetelmien toteutuksen hankaluus ja toisaalta pelien sääntöjen tuomat vaatimukset.

CCS-luokitus: **• Computing methodologies~Motion path planning • Applied computing~Computer games**

Avainsanat – Nyckelord – Keywords

reitinetsintä, etsintäavaruudet, tietokonepelit

Säilytyspaikka – Förvaringställe – Where deposited

Muita tietoja – Övriga uppgifter – Additional information

Sisältö

1 Johdanto

Monissa tietokonepeleissä on pelihahmoja, jotka liikkuvat pelialueella pelin sääntöjen mukaan. Yleisesti ottaen pelihahmon täytyy voida liikkua mistä tahansa pelialueen pisteestä, joissa pelihahmon on luvallista sijaita, mihin tahansa pelialueen pisteeseen, johon pelihahmon on luvallista siirtyä. Muut alueet tulee väistää. Yleensä on myös toivottavaa, että pelihahmon reitti on ns. optimaalinen. Optimaalisuudella tarkoitettaan yleensä lyhyintä mahdollista reittiä, mutta pituuden lisäksi reitin optimaalisuuteen voi koostua muista tekijöistä, kuten helppoudesta ja turvallisuudesta. Pelin säännöt vaikuttavat lopulta siihen, mitä reitin optimaalisuuden käsite sisältää. Optimaalisen reitin löytämisen lisäksi reitinhaun tulee olla tarpeeksi nopeaa, jotta se ei haittaa pelin sujuvuutta.

Pelialueesta on muodostettava ns. etsintäavaruus, ennen kuin reitinhaku voidaan suorittaa. Etsintäavaruuksia on monenlaisia, ja erityyppiset etsintäavaruuden sopivat erilaisiin pelialueisiin. Perinteisesti käytetty etsintäavaruus on reittipisteverkko (engl. waypoint graph), joka yksinkertaisimmillaan koostuu solmuista ja näitä yhdistävistä kaarista, joilla on pituus. Reittipisteverkko on edullinen, koska monet etsintämenetelmät toimivat siinä suoraan, mutta myös rajoittunut. Navigaatioverkko (engl. navigation mesh, navmesh) on vaihtoehtoinen tapa määrittää etsintäavaruus [Sno00, DB06]. Sen avulla etsintäavaruus pystytään määrittelemään yksityiskohtaisemmin alueena pelkkien siirtymien sijaan.

Peleissä reitinhakemiseen käytetään hyvin usein Djikstran menetelmästä kehiteltyä A*-algoritmia (lausutaan a-tähti, engl. A-star) [Dij59, Stou00]. A*-algoritmi on muodostunut käytännössä standardiksi sen monipuolisuuden, muokattavuuden ja toteutuksen yksinkertaisuuden vuoksi. A*-algoritmi toimii suoraan reittipisteverkossa.

Pro gradu -tutkielmassa perehdytään hyviksi todettuihin ja usein käytettyihin reitinetsintämenetelmiin. Siinä esitellään A*-reitinhakualgoritmi ja se kuinka sitä voidaan käyttää pelihahmojen reitinhakuun reittipisteverkossa. Seuraavaksi esitellään

navigaatioverkkomalli ja sen ominaisuuksia sekä muutamia menetelmiä, joilla siitä voidaan muodostaa reittipisteverkko. Lopuksi esitellään lyhyesti erilaisia menetelmiä reitinhaun nopeuttamiseen.

2 Pelialue ja reitinhakeminen

Pelialue (engl. level) on alue, jossa pelihahmot sijaitsevat ja toimivat [MJ08]. Pelialuetta kutsutaan yleisesti myös nimillä kenttä ja taso. Pelialue sisältää monenlaista oleellista tietoa pelialueen ominaisuuksista, jotka vaikuttavat pelihahmojen toimintaan sekä alueen ja pelitilanteen visualisointiin. Yleensä pelialue sisältää tietoa esteistä, vaikeakulkuisista ja vaarallisista paikoista, alueen sisään- ja uloskäynneistä sekä pelkästään visualisointiin vaikuttavista seikoista kuten valaistuksesta ja pelialueen pintakuvioinneista.

2.1 Pelialueen ominaisuudet

Kaikki pelialueella sijaitseva tieto sijoitetaan koordinaatistoon, jonka akselit ovat kohtisuorassa toisiaan. Yleensä pelialue on taso, jolloin sen määrittelemiseen tarvitaan kaksi akselia. Jos pelialue huomioi myös korkeuden, tarvitaan kolmas akseli. Tässä tutkielmassa akselit ovat nimetty niin että X- ja Y-akselit määrittelevät peilialueen pituuden ja leveyden ja Z-akseli korkeuden.

Pelialueita on monen kokoisia ja muotoisia. Muodon määrittelee pelin suunnittelu. Pelin säännöt voivat vaatia tietynlaista muotoa, ja toisaalta muodon voi myös määritellä käytännölliset ja esteettiset seikat. Samat tekijät vaikuttavat myös pelialueen kokoon, mutta usein pelilaitteen ominaisuudet, kuten muistin määrä, määrittävät alueen enimmäiskoon.

Monesti pelit koostuvat useista pelialueista. Pelialue jaetaan osiin monista eri syistä, joista yksi yleisimmistä on pelilaitteen muistirajoitukset. Iso pelialue ei mahdu kerralle pelilaitteen keskusmuistiin, jolloin se jaetaan pienempiin osiin ja nämä osat ladataan tarpeen mukaan, kun pelaajan pelihahmo siirtyy alueelta toiselle. Toinen yleinen pelialueen jakoperuste on pelialueen visualisointiin, eli piirtoon, liittyvät rajoitukset. Pelilaite ei välttämättä pysty piirtämään riittävän nopeasti kovin suurta määrää pelialueen yksityiskohtia. Tällöin pelialueen jakaminen osiin pienentää kerralla näkyvien yksityiskohtia määrää. Kolmas yleinen pelialueen jakoperuste on

tuotannollinen syy. Pienempiä alueita on helpompi työstää, ja työ helpompi jakaa useammalla ihmiselle. Tässä tutkielmassa keskitytään tarkastelemaan sitä, miten pelihahmot toimivat yhdellä pelialueella.

2.2 Pelihahmo

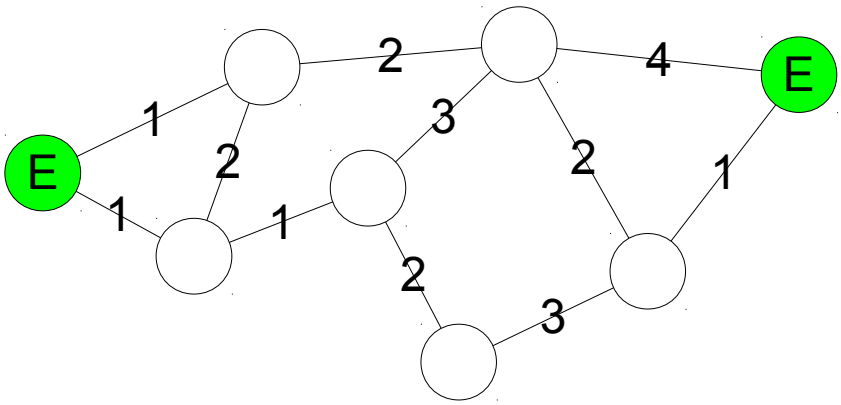
Pelihahmot ovat pelialueella dynaamisesti liikkuvia ja toimivia agentteja [FG97]. Pelihahmot toimivat pelialueella noudattaen sen sääntöjä. Pelihahmoilla on yleensä omat tavoitteensa, joihin ne pyrkivät pääsemään. Tavoitteet voivat olla hyvin yksinkertaisia tai monimutkaisia riippuen pelistä ja sen säännöistä. Päästäkseen tavoitteisiinsa pelihahmot yleensä liikkuvat pelialueella sekä ovat vuorovaikutuksessa pelialueen ja toisten pelihahmon kanssa.

Pelihahmoilla on yleensä ominaisuuksia, jotka ohjaavat niiden toimintaa ja vuorovaikutusta. Tällaisia ominaisuuksia ovat esimerkiksi liikkumisnopeus, koko ja tavoite. Joissakin peleissä pelihahmon koko on erityisen tärkeä ominaisuus reitinhaun kannalta, koska iso pelihahmo ei välttämättä mahdu liikkumaan pelialueen kapeasta kohdasta siinä missä pienempi pelihahmo mahtuu. Toisaalta joissakin peleissä pelihahmon koko on täysin toissijainen ominaisuus, jolloin reitinhaku voidaan suorittaa olettaen, että pelihahmot olevat pistemäisiä. Koko voidaan määritellä esimerkiksi hahmon ympärille asetellut kapselin kokoa kuvaavana ominaisuutena, esimerkiksi ympyrän säteenä. Ympyrän säde on yleinen valinta, koska se on yksinkertainen määrittää ja sitä on verrattain helppo käsitellä matemaattisesti, sekä se on usein riittävän tarkka arvio pelihahmon todellisesta koosta. Tarpeen vaatiessa pelihahmon koko voidaan määritellä myös muunlaisien geometristen kappaleina, kuten suorakulmiona tai monikulmiona. Yksinkertaisuuden vuoksi tässä tutkielmassa keskitytään käsittelemään pääasiassa pistemäisiä pelihahmoja.

2.3 Reitin hakeminen pelialueella

Pelialue ei sellaisenaan sovellu hyvin reitinhakemiseen, koska se sisältää paljon reitinhaun kannalta epäoleellista tietoa eikä soveltuva tieto välttämättä ole muodossa, joka olisi reitinhaun kannalta tehokkaasta hyödynnettävissä. Tästä syystä pelialueelle usein lisätään tietoa, jota käytetään pääasiassa reitinhakemiseen. Tämä tieto muodostetaan joko käsin tai automaattisesti ja tallennetaan sopivaan tietorakenteeseen.

Usein reitinhaussa käytetty tietorakenne on verkko. Verkon solmut vastaavat sallittuja sijainteja ja kaaret sallittuja siirtymiä sijaintien välillä. Solmuihin tallennetaan ainakin solmun sijainti pelialueen koordinaatistossa, ja kaariin voidaan tallentaa siirtymän pituus, joka voidaan käsittää yleisemmin siirtymän edullisuutena. Solmut, joita kaari yhdistää, sanotaan naapureiksi. Tällaista verkkoa kutsutaan reittipisteverkoksi (engl. waypoint graph) ja se on perinteisesti hyvin yleisesti peleissä käytetty etsintäavaruus [Toz03]. Kuvassa 2.1 on havainnollistettu erästä reittipisteverkkoa.



Kuva 2.1: Esimerkki reittipisteverkosta. E-kirjaimella merkityt solut ovat uloskäyntejä. Kaariin on merkitty niiden pituudet, jotka eivät tässä tapauksessa vastaa geometrista pituutta.

Pelkkä etsintäavaruus ei kuitenkaan riitä reitinhakemiseen, vaan sen lisäksi tarvitsemme menetelmän, jolla reitti haetaan. Useat reittipisteverkkoon soveltuvat reitinhakumenetelmät perustuvat Dijkstran reitinhakumenetelmään ja ovat tämän erilaisia optimointeja. Eräs yleisimmin peleissä käytetty menetelmä on A*, joka optimoi Dijkstran menetelmää ohjaamalla hakua heuristiikan avulla suuntaan, jossa

maalisolmun arvellaan sijaitsevan. Reitinhakua voidaan optimoida myös käyttämällä hyväksi reittipisteverkosta löytyviä erilaisia hierarkioita sekä lisäämällä reittipisteverkkoon kaariin tietoa verkon rakenteesta, jonka avulla reitti pystytään etsimään tosiaikaisesti, kunhan verkon solmujen ja kaarien määrä pysyy sopivan kokoisena ja pelilaitteen laskentateho on tarpeeksi suuri. Yleensä tämä ei ole ongelma nykyaikaisilla pelilaitteilla, mutta laskentatehoja voidaan tarvita pelin muuhun toimintaan siinä määrin, että reitinhakua täytyy keventää. Tällaisissa tapauksissa reitit voidaan esimerkiksi laskea etukäteen taulukkoon.

2.4 Dijkstran reitinhakumenetelmä

Edsger Dijkstra esitteli vuonna 1959 menetelmän [Dij59], jolla voidaan etsiä lyhyimpien reittien pituudet verkon solmusta verkon muihin solmuihin. Tämän lisäksi menetelmällä voidaan etsiä lyhyin reitti verkon jostain solmusta verkon toiseen solmuun. Lyhyimmällä reitillä tarkoitetaan reittiä, jonka siirtymien pituuksien summa on mahdollisimman pieni. Reitti ei siis välttämättä ole geometrisesti lyhyin reitti, vaan se voi olla mutkikas ja sisältää enemmän siirtymiä kuin mitä vähiten siirtymiä sisältävän reitti. Etsintäavaruuden ominaisuudet vaikuttavat siis oleellisesti helpoimpaan reittiin. Menetelmä vaatii, että verkon kaarien pituudet ovat positiivia.

Dijkstran reitinhakumenetelmä vaatii syötteenä verkon V sekä lähtösolmun S_L, josta reittien pituuksia aletaan etsiä. Lisäksi menetelmä ottaa valinnaisena syötteenä maalisolmun S_M, jolloin menetelmä voi lopettaa etsinnän solmun S_M löydyttyä ja yksinkertaisella lisämenetelmällä muodostaa reitin solmusta S_L solmuun S_M.

Menetelmä ylläpitää kolmea tietorakennetta. Lyhyimpien pituuksien tauluun *pituus* tallennetaan lähtösolmusta kuhunkin solmuun lyhyimmän tunnetun reitin pituus. Tämä taulu toimii menetelmän paluuarvona. Tulosolmujen tauluun *edellinen* tallennetaan jokaiselle solmuille se solmu, josta solmuun pääsee lyhyintä tunnettua reittiä pitkin. Käsittelemättömien solmujen jonoon *käsittelemättömät* tallennetaan solmut, joita ei olla vielä käsitelty.

Aluksi käydään läpi verkon V jokainen solmu S_i. Asetetaan *pituus*[S_i] = 0, jos $S_i = S_L$, muutoin *pituus*[S_i] = ∞. Seuraavaksi asetetaan taulun *edellinen*[S_i] arvo määrittelemättömiksi. Lopuksi lisätään S_i jonoon *käsittelemättömät*.

Seuraavaksi menetelmä alkaa käydä läpi jonoa *käsittelemättömät.* Menetelmä etsii jonosta solmun S_{min}, jolla on pienin arvo *pituus*[S_{min}]. S_{min} poistetaan jonosta *käsittelemättömät*. Jos arvo *pituus*[S_{min}] on ääretön, tiedämme että lähtösolmusta ei ole reittiä loppuihin käsittelemättömien solmujen jonossa oleviin solmuihin ja etsintä lopetetaan. Jos taas S_M on määritelty ja $S_{min} = S_M$, etsintä voidaan lopettaa ja reitti solmusta S_L solmuun S_M voidaan muodostaa. Muutoin, etsitään käsiteltävän solmun naapurit N_i, jotka ovat vielä jonossa *käsittelemättömät*, ja käydään ne läpi. Olkoon P_{SN} kaaren pituus solmusta S_{min} solmuun N_i. Olkoon P_N pituus lähtösolmusta S_L solmuun N_i. Tällöin $P_N = pituus[S_{min}] + P_{SN}$. Jos $P_N < pituus[N]$, asetetaan *pituus*[N] arvoksi P_N ja asetetaan *edellinen*[N_i] arvoksi S_{min}. Algoritmi 2.1 esittelee Dijkstra reitinhakumenetelmän näennäisohjelmointikielellä.

Jos solmu S_M on määritelty ja *pituus*[S_M] ≠ ∞, reitti tähän solmuun voidaan muodostaa seuraavalla tavalla. Olkoon *reitti* lista solmuja ja S käsiteltävä solmu. Asetetaan $S = S_M$. Jos arvo *edellinen*[S] on määritelty, lisätään S listan *reitti* alkuun. Tämän jälkeen asetetaan S = *edellinen*[S] ja toistetaan tarkastelu, että *edellinen*[S] on määritelty. Näin jatketaan kunnes *edellinen*[S] ei ole määritelty, jolloin lista *reitti* sisältää reitin solmusta S_L solmuun S_M.

```
Taulu EtsiReittienPituudet(Verkko verkko, Solmu lähtösolmu, Solmu maalisolmu) {
    Taulu pituus = uusi Taulu;
    Taulu edellinen = uusi Taulu;
    Jono käsittelemättömät = uusi Jono;
    iteroi (Solmu solmu : verkko.solmut()) {
        pituus[solmu] = ∞;
        edellinen[solmu] = null;
        käsittelemättömät.lisää(solmu);
    }
    pituus[lähtösolmu] = 0;

    toista (käsittelemättömät.koko() > 0) {
        Solmu pienin = käsittelemättömät.haePienin(pituus);
        jos (maalisolmu != null && pienin == maalisolmu) {
            palauta pituus;
        }
        käsittelemättömät.poista(pienin);
        jos (pituus[pienin] == ∞) {
            palauta pituus;
        }
        Lista naapurit = verkko.naapurit(pienin);
        iteroi (Solmu naapuri : naapurit) {
            Pituus pituus = pituus[pienin] + verkko.kaarenPituus(pienin, naapuri);
            jos (pituus < pituus[naapuri]) {
                pituus[naapuri] = pituus;
                edellinen[naapuri] = pienin;
            }
        }
    }
    palauta pituus;
}
```

Algoritmi 2.1 Djikstran menetelmä näennäisohjelmointikielellä kirjoitettuna.

2.5 Reitinhaku A*-menetelmällä

A* on vanha reitinhakumenetelmä. Se kehitettiin 1968 ja on edelleen erittäin suosittu ja yleisesti käytetty menetelmä. A* menetelmää käytetään reitinhaun lisäksi ratkaisemaan monen muunlaisia ongelmia. Oleellista on, että ongelmasta voidaan muodostaa reittipisteverkoston kaltainen tilaverkko, joka voidaan ajatella olevan reittipisteverkon yleistys. A*-menetelmää käytetään usein peleissä [Stou00].

A*-menetelmä perustuu Dijkstran reitinhakumenetelmään. Toisin kuin Dijkstan reitinhakumenetelmä, A*-menetelmä etsii vain reittipisteverkon lyhyimmän reitti lähtö- ja maalipisteen välille eikä siis reittien pituuksia verkon muihin solmuihin. A*-menetelmä käyttää niin sanottua paras ensin -hakumenetelmää, jossa lupaavimmat solmut tutkitaan ensin. Lupaavimmat solmut päätellään heuristisesti arvioimalla kunkin solmun kautta kulkevan reitin kokonaispituus ja päivittämällä tätä tietoa tarpeen mukaan etsinnän aikana.

A*-menetelmä pitää yllä kahta tietorakennetta, joihin tallennetaan ja joista poistetaan solmuja haun edetessä. Ensimmäisessä tietorakenteessa pidetään avoimia soluja ja toisessa suljettuja solmuja. Selkeiden ja yksinkertaisuuden vuoksi nimitetään näitä tietorakenteita avoimien ja suljettujen listoiksi, vaikka varsinaiset tietorakenteet eivät olisikaan listoja. Haun aikana avoimien listassa pidetään solmuja, jotka on löydetty mutta joissa ei olla vielä vierailtu. Suljettujen listassa pidetään solmuja, joissa on vierailtu. Jokaiseen löydettyyn solmuun tallennetaan lisäksi sen reitin pituus, joka johti tähän solmuun, sekä arvio pituudesta, joka tästä solmusta on maaliin, sekä tulosolmu eli solmu, josta tähän solmuun tultiin. Lähtöpisteestä solmuun johtavan reitin pituuden funktiota kutsutaan kirjallisuudessa nimellä g(x), jossa x on tarkasteltava solmu. Funktiota, joka laskee arvion jäljellä olevan reitin pituudelle, kutsutaan nimellä h(x). Tämän funktion tulee olla luvallinen, eli se ei saa koskaan yliarvioida jäljellä olevaa matkaa. Jos h(x) ei ole luvallinen, niin löydetty reitti ei välttämättä ole optimaalinen. Solmun kautta kulkevan reitin kokonaispituuden arvion laskevaa funktiota kutsutaan nimellä f(x). Funktio f(x) saadaan kaavalla g(x) + h(x) ja sen arvoa käytetään etsinnän ohjaamiseen.

Haun alussa suljettujen lista on tyhjä ja avoimien listassa on vain lähtösolmu. Menetelmä käy läpi avoimien listaa ottamalla sieltä joka kierroksella lupaavimman solmun. Lupaavin solmu on solmu, jolla on pienin f(x). Lupaavin solmu lisätään aina suljettujen listaan. Jos solmu on maali, haku lopetetaan ja reitti muodostetaan etenemällä tulosolmujen ketjua, kunnes lähtösolmu on saavutettu. Jos taas solmu ei ole maalisolmu, solmun jokainen naapurisolmulle lasketaan g(x), h(x) ja f(x). Jos naapurisolmu ei ole avoimien listassa, se lisätään sinne. Jos naapurisolmu on avoimien tai suljettujen listassa ja g(x) on pienempi kuin aikaisempi arvo, naapurisolmun arvot lasketaan uudelleen, tulosolmuksi asetetaan nykyinen solmu ja naapurisolmu poistetaan suljettujen listalta, jos se oli siellä. Jos avoimien solmujen lista tyhjenee, ennen kuin maalisolmu on saavutettu, haku lopetetaan ja todetaan, ettei reittiä löytynyt. Algoritmi 2.2 esittelee A*-menetelmän yleisellä tasolla näennäisohjelmointikielellä kirjoitettuna.

```
Reitti EtsiReitti(Verkko verkko, Solmu lähtösolmu, Solmu maaliSolmu, Heuristiikka heuristiikka) {
    Lista avoimet = uusi Lista;
    Lista suljetut = uusi Lista;
    avoimet.lisää(lähtösolmu);
    toista (avoimet.koko() > 0) {
        Solmu lupaavin = avoimet.etsiPieninPituus();
        jos (lupaavin == maaliSolmu) {
            Reitti reitti  = rakennaReitti(maaliSolmu);
            palauta reitti;
        }
        suljetut.lisää(lupaavin);
        Lista naapurit = verkko.naapurit(lupaavin);
        iteroi (Solmu naapuri : naapurit) {
            Luku pituusNaapuriin = lupaavin.pituus()
                    + verkko.kaarenPituus(lupaavin, naapuri);
            jos (pituusNaapuriin < naapuri.pituus()) {
                jos (avoimet.sisältää(naapuri)) {
                    avoimet.poista(naapuri);
                }
                jos (suljetut.sisältää(naapuri)) {
                    suljetut.poista(naapuri);
                }
            }
            jos (avoimet.eiSisällä(naapuri) ja suljetut.eiSisällä(naapuri)) {
                naapuri.asetaPituusTähän(pituusNaapuriin);
                avoimet.lisää(naapuri);
                naapuri.asetaPituus(pituusNaapuriin
                        + heuristiikka.arvioiPituus(naapuri, maaliSolmu));
                naapuri.asetaTulosolmu(lupaavin);
            }
        }
    }
    palauta tyhjä;
```

Algoritmi 2.2 A*-menetelmä näennäisohjelmointikielellä kirjoitettuna.

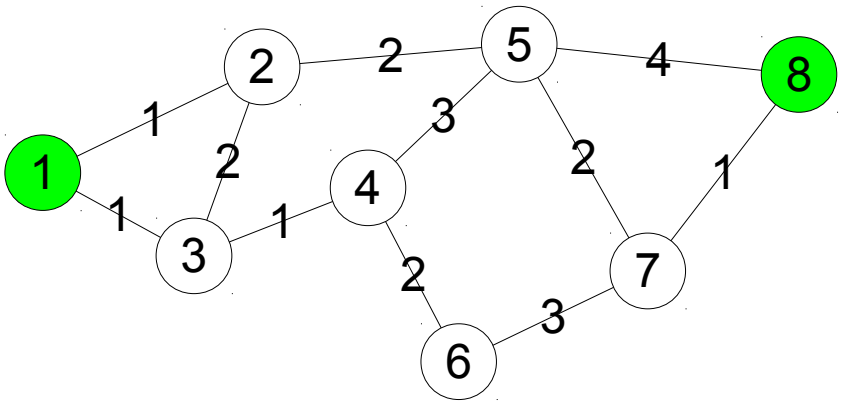
A*-menetelmällä on monia edullisia ominaisuuksia. Ensinnäkin, se löytää reitin lähtösolmusta maalisolmuun, jos tällainen reitti on etsintäavaruudessa olemassa. Toiseksi, muodostettu reitti on aina optimaalinen, jos funktio h(x) on luvallinen eli h(x) ei koskaan yliarvioi jäljellä olevaa matkaa. Kolmanneksi, A*-menetelmä käyttää tehokkaimmin hyväkseen käytettyä heuristiikkaa kuin mikään muu etsintämenetelmä, eli A*-menetelmä tutkii vähiten solmuja löytääkseen optimaalisen reitin. Kaikki nämä ominaisuudet on todistettu [RN03].

A*-algoritmi on myös muokkautuva. Funktiota g(x) ja h(x) voidaan muokata ja laajentaa, niin että ne ottavat huomioon muitakin seikkoja kuin pelkät reittipisteverkon kaarien pituudet. Tällöin reitinetsinnästä voidaan tehdä esimerkiksi taktisempaa [Ste02].

2.6 Reitinhaku siirtymätaulukkoa käyttäen

Joissain tapauksissa reitin hakeminen tosiaikaisesti pelin ollessa käynnissä on epätoivottavaa. Esimerkkinä tällaisesta tilanteesta voisi olla tilanne, jossa pelilaitteen laskentateho on rajallinen tai jossa laskentatehoa tarvitaan muuhun kuin reitinhakemiseen. Ratkaisuna voisi olla reittien laskeminen etukäteen taulukkoon [Dic03].

Menetelmän perustana on kaksiulotteinen taulukko, johon tallennetaan solmujen tunnisteita. Kutsutaan tällaista taulukkoa siirtymätaulukoksi. Siirtymätaulukossa on rivi ja sarake jokaiselle etsintäavaruuden solmulle. Määritellään, että jokainen rivi vastaa lähtösolmua ja että jokainen sarake vastaa maalisolmua. Lähtösolmulla tarkoitetaan tässä yhteydessä sitä solmua, jossa sijaitsemme etsinnän sillä hetkellä. Taulukon jokaiseen solu määrittelee siirtymän, joka suoritetaan, kun haluamme päästä lähtösolmusta maaliin. Kuvat 2.2 havainnollistaa erästä reittipisteverkkoa ja kuva 2.3 havainnollistaa siitä muodostettua siirtymätaulukkoa.



Kuva 2.2: Eräs reittipisteverkko, johon on merkitty solmujen tunnisteet numeroina.

		Maalisolmu							
		1	2	3	4	5	6	7	8
Lähtö-solmu	1	1	2	3	3	2	3	2	2
	2	1	2	3	3	5	3	5	5
	3	1	2	3	4	4	4	4	4
	4	3	3	3	4	5	6	6	6
	5	2	2	2	4	5	7	7	7
	6	4	4	4	4	4	6	7	7
	7	5	5	5	5	5	6	7	8
	8	7	7	7	7	5	7	7	8

Kuva 2.3: Kuvan 3.2 reittipisteverkosta muodostettu siirtymätaulukko.

Etsintämenetelmä toimii seuraavasti. Oletetaan, että haluamme etsiä reitin solmusta A solmuun B. Aluksi asetamme lähtösolmuksi solmun A ja lisäämme sen listaan, johon koko reitti solmusta A solmuun B tallennetaan. Seuraavaksi tarkastamme, onko lähtösolmu sama solmu kuin B. Jos on, olemme löytäneet reitin, joten palautetaan se. Jos lähtösolmu ei ole B, niin etsimme taulukosta lähdesolmua vastaavan rivin ja B:tä vastaavan sarakkeen. Se solu, joka yksiselitteisesti risteää löydettyä riviä ja saraketta, sisältää solmun tunnisteen, johon meidän tulee seuraavaksi siirtyä, jotta pääsemme kohti

maalisolmua B. Asetetaan löydetty solmu lähtösolmuksi ja aloitetaan alusta tarkastamalla, onko uusi lähtösolmu sama kuin B, jne. Algoritmi 2.3 selventää etsintämenetelmää näennäisohjelmointikielellä.

```
Reitti EtsiReitti(Solmu lähtösolmu, Solmu maalisolmu) {
    Solmu lähdesolmu = lähtösolmu;
    Reitti reitti = uusi Reitti;
    reitti.lisää(lähdesolmu);
    toista (lähdesolmu != maalisolmu) {
        lähdesolmu = SIIRTYMÄTAULUKKO[lähdesolmu][maalisolmu];
        reitti.lisää(lähdesolmu);
    }
    palauta reitti
}
```

Algoritmi 2.3 Menetelmä esilasketun reitin etsimiseen taulukosta näennäisohjelmointikielellä kirjoitettuna.

Taulukko voidaan muodostaa esimerkiksi pelin käynnistyessä tai sitäkin aikaisemmin jossain sopivassa tuotannon vaiheessa kuten esimerkiksi pelin kääntämisen yhteydessä. Muodostamiseen voidaan käyttää esimerkiksi A*-menetelmää siten, että jokaisen taulun solua vastaavasta lähtösolmusta etsitään reitti solua vastaavaan maalisolmuun ja soluun tallennetaan löydetyn reitin ensimmäinen siirtymä.

Esilasketun taulun käyttö voi aiheuttaa ongelmia keskusmuistin riittävyyden suhteen. Koska taulukon koko kasvaa suhteessa n^2, jossa n on solmujen määrä, solmumäärän kasvaessa taulukko voi vaatia liikaa keskusmuistia. Eräs ratkaisu tähän on menetelmän laajennos, jossa etsintäavaruutta jaetaan pienempiin osiin ja kustakin osasta muodostetaan taulukot, jotka sisältävät vain kyseisen osan solmuja [Dic03]. Näiden taulukkojen lisäksi muodostetaan taulukko, joka sisältää vain alueita yhdistäviä solmuja. Kutsutaan tällaisia solmuja rajasolmuiksi. Jos lähtö ja maali solmut sijaitsevat samalla alueella, etsintä tapahtuu kuten aiemmin. Jos solmut ovat eri alueilla, etsintä suoritetaan kolmessa vaiheessa. Ensiksi etsitään reitti lähtösolmusta sopivaan lähtöalueen rajasolmuun, tämän jälkeen lähtöalueen rajasolmusta maalialueen rajasolmuun ja

viimeiseksi etsitään reitti maalialueen maalisolmuun. Löydetyt reitit yhdistämällä saamme lopullisen reitin. Ongelmana tässä optimointimenetelmässä on löytää optimaalinen jako alueiden välillä.

Esilasketussa taulukossa on muitakin ongelmia. Taulukkoa käytettäessä menetetään kaikki etsintäavaruuden dynaamisuus. Jos nimittäin pelin aikana etsintäavaruuden solmuja poistuu, niitä lisätään tai kaarien pituuden muuttuvat, taulukko ei ole enää kelvollinen. Tällöin taulukko tulisi muodostaa uudelleen. Jos etsintäavaruus muuttuu usein, taulukon uudelleenmuodostaminen ei ole välttämättä kelvollinen ratkaisu, koska taulukon tuoma hyöty voidaan menettää.

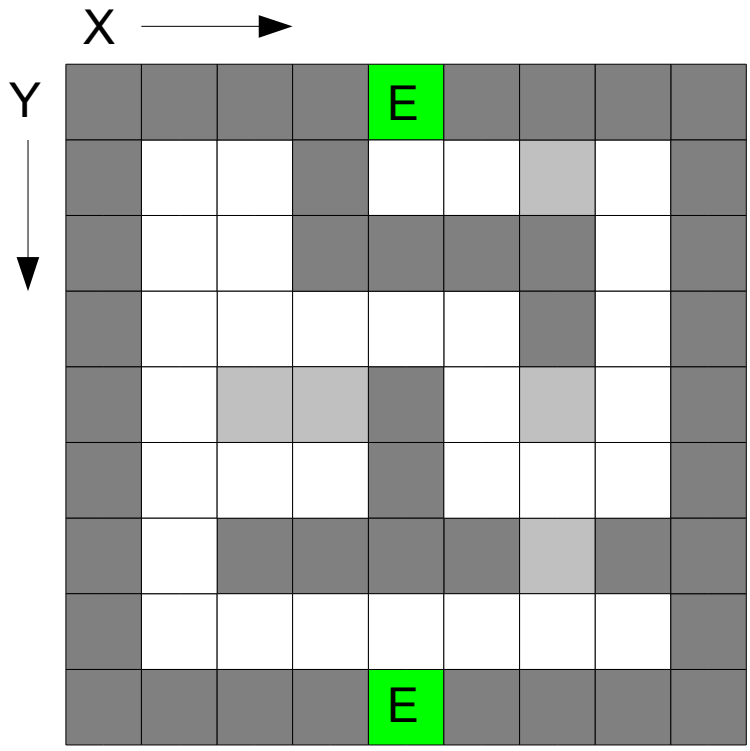
3 Pelialueiden etsintäavaruuksia

Tässä luvussa esitellään usein käytettyjä pelialuetyyppejä ja niihin liittyviä etsintäavaruuksia. Huomattavaa on, että kaikki esimerkit käyttävät reittipisteverkkoa etsintäavaruutena vaikka pelialueen muodostamiseen käytetyt periaatteet ovat hyvinkin erilaisia. Oletamme myös, että reitinhakuun käytetään A*-menetelmää, jolloin reittipisteverkko on ainoa sopiva vaihtoehto. Reittipisteverkon tekniset toteutukset voivat vaihdella.

3.1 Ruudukkopohjainen pelialue

Ruudukkopohjainen pelialue on suorakulmion muotoinen alue, joka koostuu ruuduista, jotka ovat kaikki saman kokoisia neliöitä ja jotka ovat kiinni toisissaan kohtisuoraan niin että jokaisella ruutu on sivuistaan kiinni korkeintaan neljässä muussa ruudussa ja kulmistaan korkeintaan kahdeksassa muussa. Lisäksi jokainen ruutu jakaa vähintään kaksi sivua muiden ruutujen kanssa ja kulman vähintään kolmen ruudun kanssa. Toisissaan kiinni olevia ruutuja kutsutaan naapuriruuduiksi. Shakkilauta on klassinen ruudukkopohjainen pelialue. Ruutu on tällaisen pelialueen muodostamisessa käytetty pienin osanen ja kaikki pelialueen tieto sijoitetaan tarpeen mukaan kuhunkin ruutuun ruudun tarkkuudella. Voidaan ajatella, että ruudut ovat pelialueen koordinaatiston pienin ja ainoa tarjolla oleva tarkkuus tallentaa tietoa. Koordinaatistossa on X- ja Y-akselit, jotka ovat kohtisuorassa toisiaan nähden. X-koordinaatti on vaakatasossa ja arvot kasvavat oikealle mentäessä ja Y-koordinaatti on pystytasossa ja sen arvot kasvavat alaspäin mentäessä. Origon voi sijaita periaatteessa missä vain, mutta käytännön syistä se sijaistee yleensä alueen vasemmassa yläkulmassa tai alueen keskellä. Kuvassa 3.1 on havainnollistettu erästä ruutupohjaista pelialuetta.

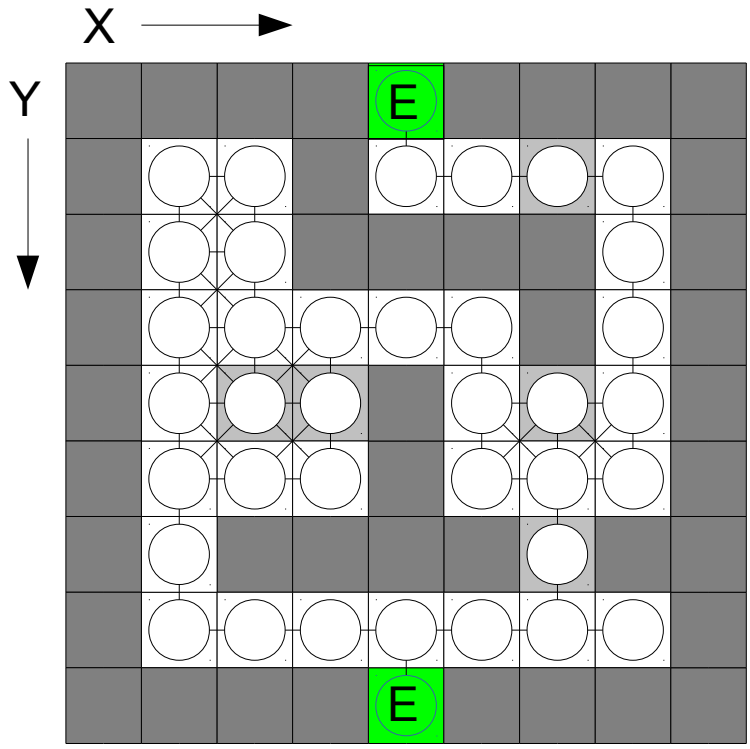
Ruudukkopohjainen pelialue on perinteisesti paljon käytetty pelialue tyyppi, jota käytetään erityisesti silloin, kun peli on säännöiltään yksinkertainen tai kun pelilaite on erityisen rajoittunut muistiltaan tai suorituskyvyltään. Pelialue voidaan tallentaa helposti kaksiulotteiseen taulukkoon, jossa jokainen solu vastaa yhtä pelialueen ruutua. Ruudut sijoitellaan taulukkoon niin, että niiden ja vastaavien solujen naapurit ovat samat. Taulukkoa on nyt helppo selata, ja ruudun naapurit löytyvät helposti taulukon osoittimia



Kuva 3.1: Kuvassa esimerkki ruutupohjaisesta pelialueesta. Tumman harmaat ruudut ovat läpipääsemättömiä ruutuja, vaalean harmaan hankalakulkuisia ruutuja, vaaleat helppokulkuisia ruutuja. E-ruuduista pääsee siirtymään muihin pelialueisiin.

muokkaamalla. Esimerkiksi ruudun, joka sijaistee koordinaatissa (x, y), pohjoisessa sijaitseva naapuri löytyy koordinaatista (x, y – 1), oikealla sijaitseva naapuri koordinaatista (x + 1, y), jne.

Ruudukkopohjaisessa pelialueessa etsintäavaruus muodostuu luontevasti sijoittamalla reittipisteverkon solmut ruutuihin. Kaaret muodostetaan sellaisten solmujen välille, joita vastaavat ruudut ovat toistensa naapureita ja joiden välillä on sallittua siirtyä. Solmujen pituudeksi voidaan määritellä geometrinen pituus, jota painotetaan arvolla, joka vastaa ruutuun liikkumisen hankaluutta. Kuva 3.2 havainnollistaa etsintäavaruutta, joka on muodostettu kuvan 3.1 mukaisesta pelialueesta.



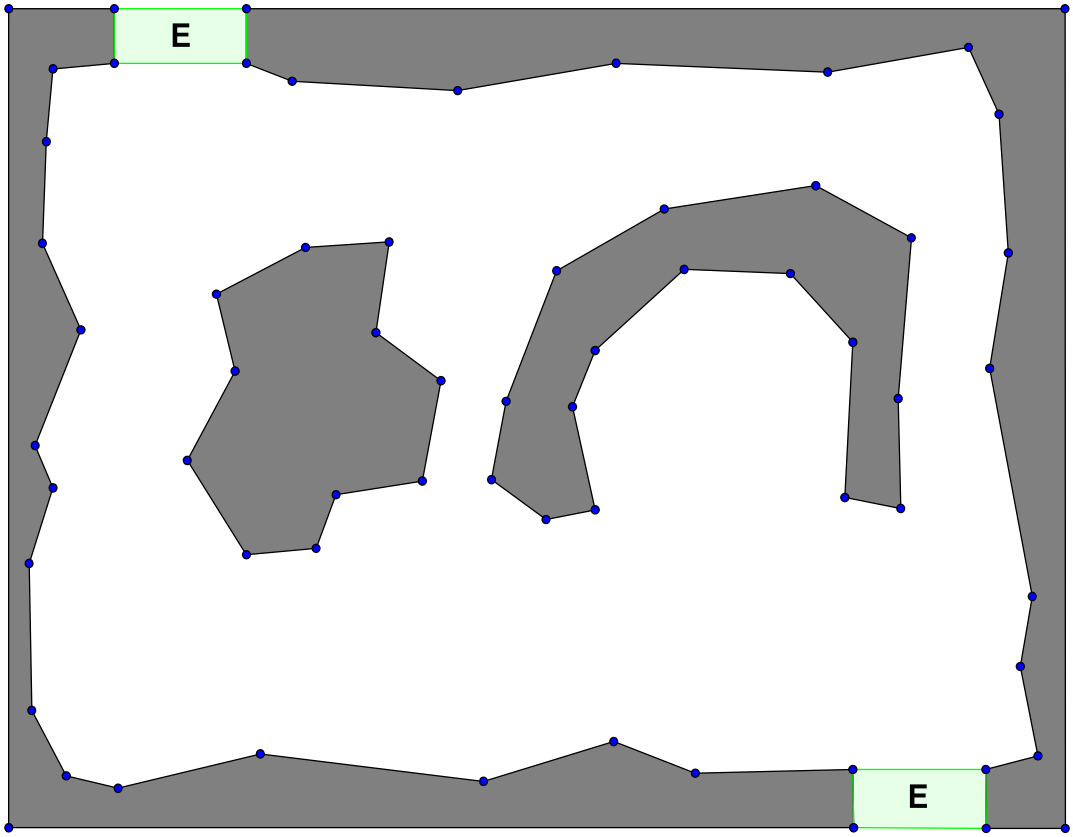
Kuva 3.2: Kuvassa esimerkki etsintäavaruudesta, joka on muodostettu ruudukkopohjaisesta pelialueesta. Tilan puutteen vuoksi kaarien pituuksia ei ole merkitty. Harmailla alueilla sijaitseviin solmuihin kiinnittyvät kaaret ovat pidempiä

Neliöiden sijaan ruudukkopohjainen pelialue voi muodostua myös muunlaisista säännöllisistä muodoista [Yap02]. Riittää että ruudut ovat yhdenmuotoisia ja -kokoisia ja että ne voidaan latoa vierekkäin niin, että pelialue peittyy. Eräs usein käytetty vaihtoehto neliölle on kuusikulmio. Kuusikulmiosta koostuva pelialue poikkeaa nelikulmioista koostuvasta pelialueesta erityisesti siten, että solmuilla on enintään kuusi naapuria. Tällöin myös jokainen solu jakaa sivun naapurinsa kanssa.

3.2 Vapaamuotoinen pelialue

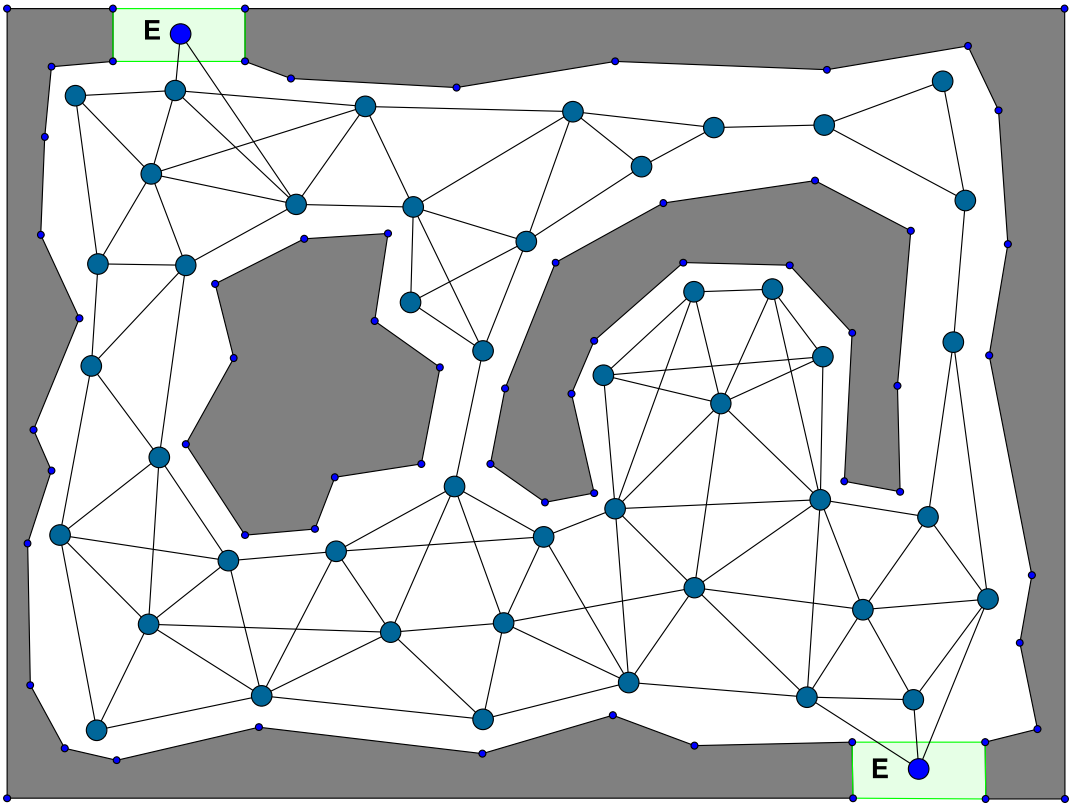
Vapaamuotoinen pelialue on pelialue, jossa pelialueen sisältö on sijoiteltu vapaasti pelialueelle ilman erityisiä rajoituksia. Toisin kuin ruudukkopohjaisessa pelialueessa, jossa sisältö sijoiteltiin aina ruudun tarkkuudella, vapaamuotoisessa pelialueessa sisältö voidaan asetella vapaasti mihin pelialueen pisteeseen tahansa. Esineet voivat olla hyvin erimuotoisia ja -kokoisia. Tällainen pelialuetyyppi on nykyisin erittäin yleinen erityisesti peleissä, joissa on pyritty tuottamaan laadukasta, vaihtelevaa ja runsasta

sisältöä ja jotka usein pyrkivät luomaan kuvan todenmukaisesta pelimaailmasta. Todenmukaisen pelialueen luominen ruudukkopohjaiselle pelialueelle on usein hyvin haastavaa, koska sisältö tulee asetella ruutujen tarkkuudella. Monet modernit pelilaitteet pystyvät myös käsittelemään monipuolisia pelialueita. Pelilaitteiden laskentatehojen ja muistin määrän kasvu on myös mahdollistanut monipuolisemmat pelialueet. Kuva 3.3 havainnollistaa erästä yksinkertaistettua vapaamuotoista pelialuetta.



Kuva 3.3: Vapaamuotoisessa pelialueessa esteet, osittaiset esteet, esineet ja poistumisreitit voivat sijaita missä vain pelialueella. Lisäksi ne voivat olla hyvin vapaamuotoisia.

Vapaamuotoisen pelialueen etsintäavaruuden solmut kannattaa sijoitella vapaasti. Solmuja ei kannata sijoitella ruudukkopohjaisesti, sillä silloin solmut eivät välttämättä myötäilisi esteiden muotoja kovin hyvin. Huonosti asetellut solmut voisivat johtaa tilanteisiin, joissa kaaret leikkaavat esteitä, mikä en epätoivottavaa. Tämä nimittäin voisi johtaa siihen, että pelihahmo liikkuisi esteen läpi siirtyessään kaarta pitkin solmusta toiseen. Kuvassa 3.4 havainnollistetaan erästä reittipisteverkkoa, joka on muodostettu käsin kuvan 3.3 pelialueelle.



Kuva 3.4: Esimerkki reittipisteverkosta, joka on muodostettu kuvan 3.3 pelialueeseen.

Vaikkakin reittipisteverkko on yleisesti peleissä käytetty etsintäavaruustyyppi, se ei ole ongelmaton. Ensimmäinen ongelma on reittipisteiden sijoittaminen pelialueelle niin, että ne kattavat riittävän osan kulkukelpoisesta alueesta ilman, että reittipisteiden määrä kasvaa kovin suureksi. Paraskin reitinhakualgoritmi voi hidastua reittipisteiden määrän kasvaessa, mikä voi aiheuttaa ongelmia erityisesti tosiaikaisissa peleissä, joissa pelin tai pelihahmojen pysähtyminen reitinhaun ajaksi on usein epätoivottavaa. Reittipisteet tulee myös sijoitella niin, että pelihahmojen liikkuminen niiden kautta vaikuttaa järkevältä toiminnalta. Koska pelihahmot liikkuvat aina etsintäavaruuden kaaria pitkin solmusta toiseen, kaaret eivät myöskään saisi leikata esteitä. Muutoin syntyy helposti vaikutelma, että pelihahmo siirtyy esteen läpi.

Reittipisteverkon solmut on aseteltava käsin jokaiseen pelialueeseen. Solmujen asetteleminen voi olla aikaa vievää, eikä aina ole selvää, mihin kukin solmu kannattaa asettaa ja mitkä solmut kannattaa yhdistää kaarella. Asetteleminen on myös subjektiivista, joten asettelijalla on vastuu siitä, että pelihahmot löytävät verkosta luonnollisia reittejä.

Ongelmia aiheuttaa myös pelihahmojen yhtäaikainen liikkuminen sekä esineiden väistö reittipisteverkossa. Hahmot saattavat liikkua yhtä aikaa samaa tai vierekkäistä kaarta pitkin niin, että pelihahmot törmäävät toisiinsa. Jotkin pelit sallivat hahmojen liikkumisen toistensa läpi, jolloin törmäykset eivät ole ongelmia. Joissain peleissä tällainen läpisiirtyminen ei ole toivottavaa, ja se tulee estää käytettävissä olevilla keinoilla, kuten dynaamisesti siirtämällä pelihahmoja niin, etteivät ne kiertävät toisensa. Reittipisteverkkoa käytettäessä tällaisen väistämisen toteuttaminen virheettömästi on erittäin hankalaa, koska se ei tarjoa tarpeeksi tietoa liikkumiseen soveltuvasta alueesta. Koska tiedossa on vain solmut ja kaaret, niiltä poikkeaminen voi johtaa pelihahmon siirtymiseen pois liikkumiseen kelpaavalta alueelta. Tämä voi aiheuttaa monenlaisia ongelmia, jotka lievimmillään ovat visuaalisia mutta pahimmallaan rikkovat pelin sääntöjä pilaten pelikokemuksen. Tämänkaltaisiin ongelmiin ei ole kovin hyviä ratkaisua, jos käytössä on reittipisteverkko [Toz03].

4 Navigaatioverkkomalli

Navigaatioverkkomalli on etsintäavaruus, joka pyrkii täydentämään reittipisteverkon puutteita [Sno00]. Navigaatioverkkomalli laajentaa reittipisteverkkoa muuttamalla verkon esitystapaa lisäten siihen hyödyllistä tietoa reitinetsintäavaruudesta. Sallittujen siirtymien lisäksi navigaationverkkomalli määrittelee alueet, joilla on sallittua liikkua. Navigaatioverkkomalli voidaan muodostaa käsin käyttäen mitä tahansa työkalua, jolla voidaan muodostaa ja muokata komiulotteisia malleja. Vaihtoehtoisesti navigaationverkkomalli voidaan muodostaa automaattisesti [Ham08].

Reitinetsintä navigaationverkkomallissakin suoritetaan yleensä käyttäen esimerkiksi A*-menetelmä, mutta vaihtoehtoisiakin menetelmiä on [WC02]. Koska A*- ja Djikstran menetelmä vaativat reittipisteverkon, navigaationverkkomallista täytyy muodostaa sellainen. Reittipisteverkon muodostamiseen ja lopullisen reitin hakemiseen on erilaisia menetelmiä [DB06].

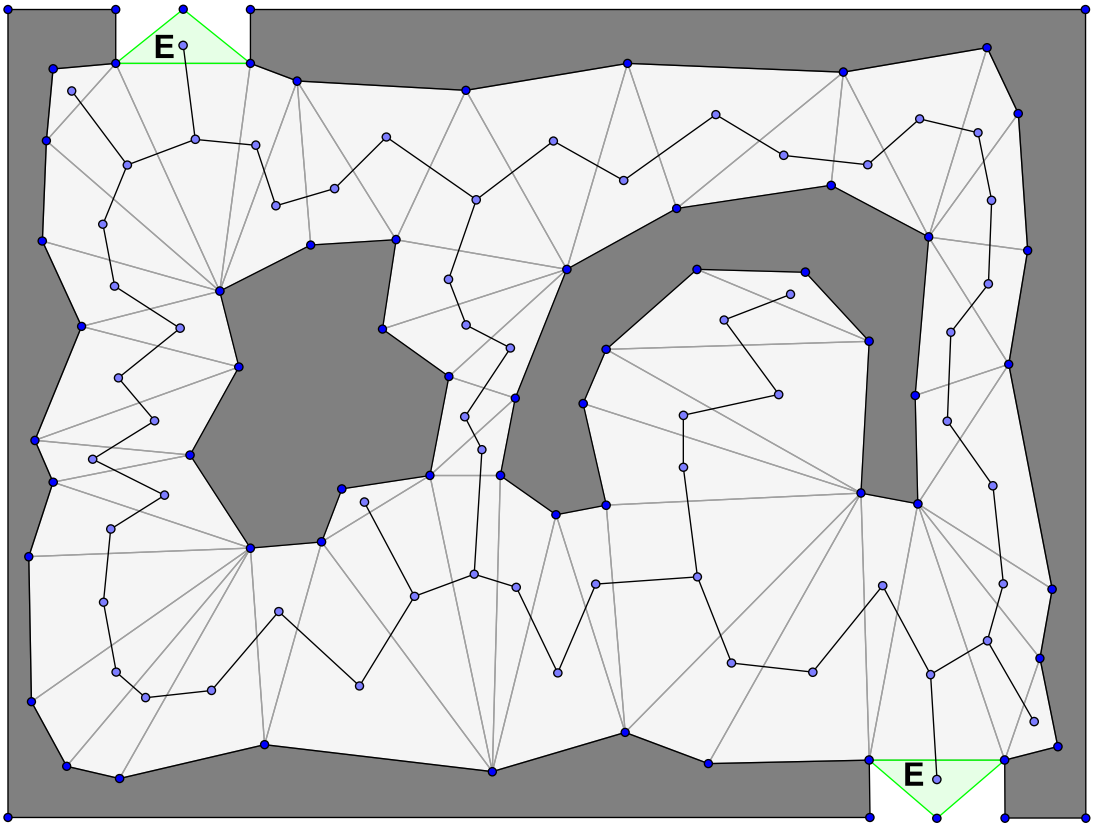
4.1 Navigaatioverkkomallin rakenne

Navigaatioverkkomalli koostuu kolmioista. Jokaisen kolmion määrittelee kolme pistettä, jotka sijaitsevat joko kaksi- tai kolmiulotteisessa avaruudessa, pelialueesta riippuen. Jokainen kolmio tunnetusti määrittelee pinnan. Navigaatioverkkomallissa tämä pinta tulkitaan alueeksi, jossa pelihahmon on luvallista liikkua. Navigaatioverkkomallin kolmioiden pinnat yhdessä määrittelevät siis kaikki ne pisteet, joissa pelihahmojen on luvallista sijaita.

Navigaatioverkkomallilla on reuna. Reuna erottaa ne alueet, joissa pelihahmojen on luvallista sijaita, niistä alueista, joissa pelihahmojen ei ole luvallista sijaita. Koska navigaatioverkkomallin muodostuu kolmioista, reunan täytyy mukailla näitä kolmioiden sivuja. Tarkalleen ottaen reunaan kuuluu kaikki ne navigaatioverkkomallin kolmioiden sivut, joita ei jaeta muiden navigaatioverkkomallin kolmioiden kanssa.

Kolmio voi tunnetusti jakaa yksi, kaksi, kolme tai ei yhtään pistettä toisen kolmion kanssa. Navigaatioverkkomallissa kolmioista voidaan siirtyä toiseen kolmioon vain ja vain jos nämä kolmio jakavat tasan kaksi pistettä keskenään, jolloin nämä kolmiot

jakavan keskenään tasan yhden sivun. Kutsutaan näitä sivuja portaaleiksi. Koska kolmion sivu muodostuu aina kahdesta pisteestä, myös portaali muodostuu täten aina kahdesta pisteestä ja se on erityisesti näin pisteitä yhdistävä jana. Kolmioita, jotka jakavat kolme pistettä toisen kolmion kanssa, ei sallita, sillä tällaiset kolmiot eivät tuo navigaatioverkkomalliin mitään oleellista lisätietoa. Kuvassa 4.1 on eräs navigaatioverkkomalli, johon portaalin jakavat kolmiot on yhdistetty painopisteistään toisiinsa. Muodostunut murtoviiva osoittaa siis sallitut siirtymät kolmioiden välillä.



Kuva 4.1: Eräs navigaatioverkkomalli. Murtoviivojen kaaret on yhdistetty kolmioiden painopisteisiin.

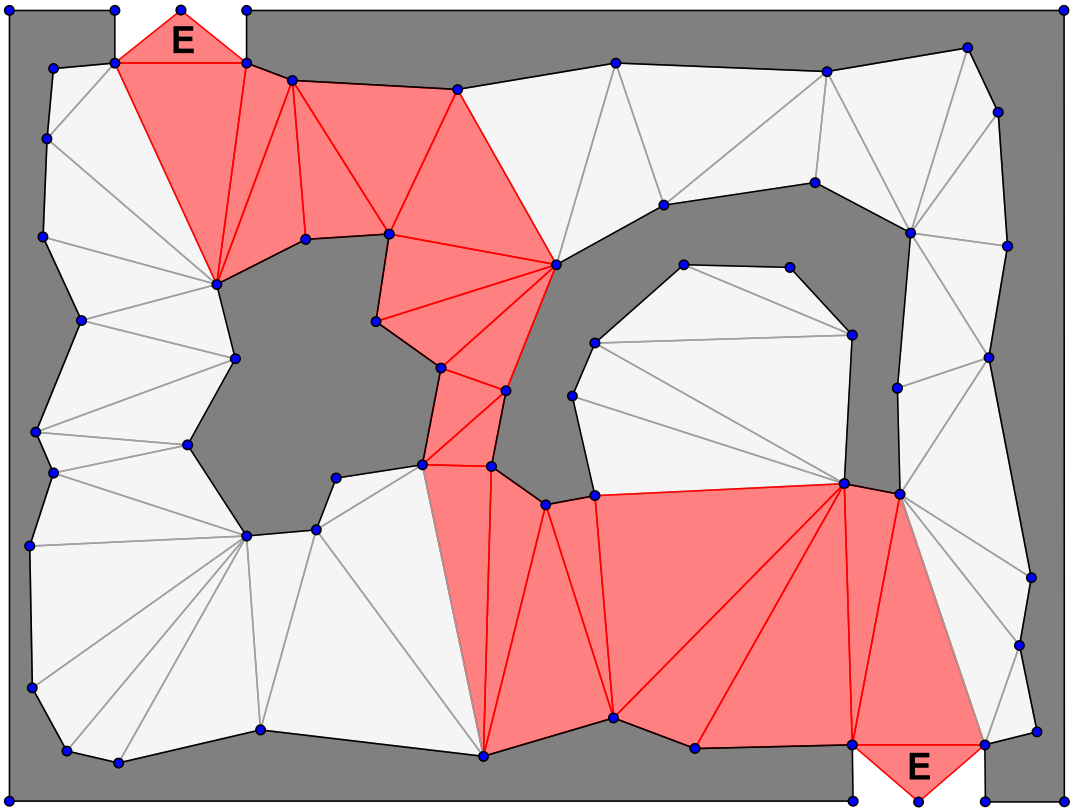
5 Reitinhaku navigaatioverkkomallissa kanavaa käyttäen

Koska pelihahmojen on luvallista sijaita vain ja ainoastaan navigaatioverkkomallin määräämällä alueella, myös pelihahmon jokaisen kelvollisen reitin täytyy sijaita kokonaisuudessaan siellä. Jos siis otamme navigaatioverkkomallin kaksi mitkä tahansa pistettä ja oletamme, että löytyy reitti, joka yhdistää nämä pisteet, niin tämän reitin täytyy kulkea ainoastaan navigaatioverkkomallin kolmioiden pintoja pitkin. Ne kolmiot, joiden kautta reitti kulkee, muodostaa navigaatioverkkomallin kolmioiden osajoukon. Tätä kolmioiden osajoukkoa kutsutaan kanavaksi [DB06]. Kun pelihahmolle etsitään reittiä, voimme ensiksi etsiä kanavan ja tämän avulla voimme muodostaa pelihahmolle lopullisen reitin.

5.1 Kanava

Kanava sisältää kaikki ne navigaatioverkkomallin kolmiot, joita pitkin pelihahmon täytyy kulkea, jotta se pääsisi lähtösolmusta maalisolmuun. Kanavaan ei kuulu muita navigaationverkkomallin kolmioita. Kun pelihahmon reittiä määritellään, kanavaa käsitellään omana erillisenä kokonaisuutena, eräänlaisena alinavigaatioverkkomallina. Siirtymät muiden kuin kanavan kolmioiden välillä ei ole sallittua. Kuva 5.1 havainnollistaa erästä kanavaa.

Kanava sisältää vähintään yhden navigaatioverkkomallin kolmion. Jos kanavassa on vain yksi kolmio, niin lähtö- ja maalipisteiden täytyy sijaita tässä kolmiossa. Jos lähtö- ja maalipisteet sijaitsevat eri kolmioissa, niin kanavassa on vähintään kaksi kolmiota, kuitenkin enintään navigaatioverkkomallin kaikki kolmiot. Koska kanava määrittelee yksikäsitteisesti ne navigaatioverkkomallin kolmiot, joiden kautta täytyy kulkea, jotta päästään lähtöpisteestä maalipisteeseen, niin kanavan jokaisella kolmiolla on korkeintaan kaksi naapuria, joihin on luvallista siirtyä. Kanava muodostaa siis eräänlaisen järjestetyn ketjun, tai linkitetyn listan, jossa voidaan siirtyä vain edelliseen tai seuraavaan kanavan kolmioon. Koska siirtyminen tapahtuu portaalien kautta, pelihahmon täytyy kulkea jokaisen kanavan sisältämän portaalin kautta kulkiessaan lähtöpisteestä maalipisteeseen.



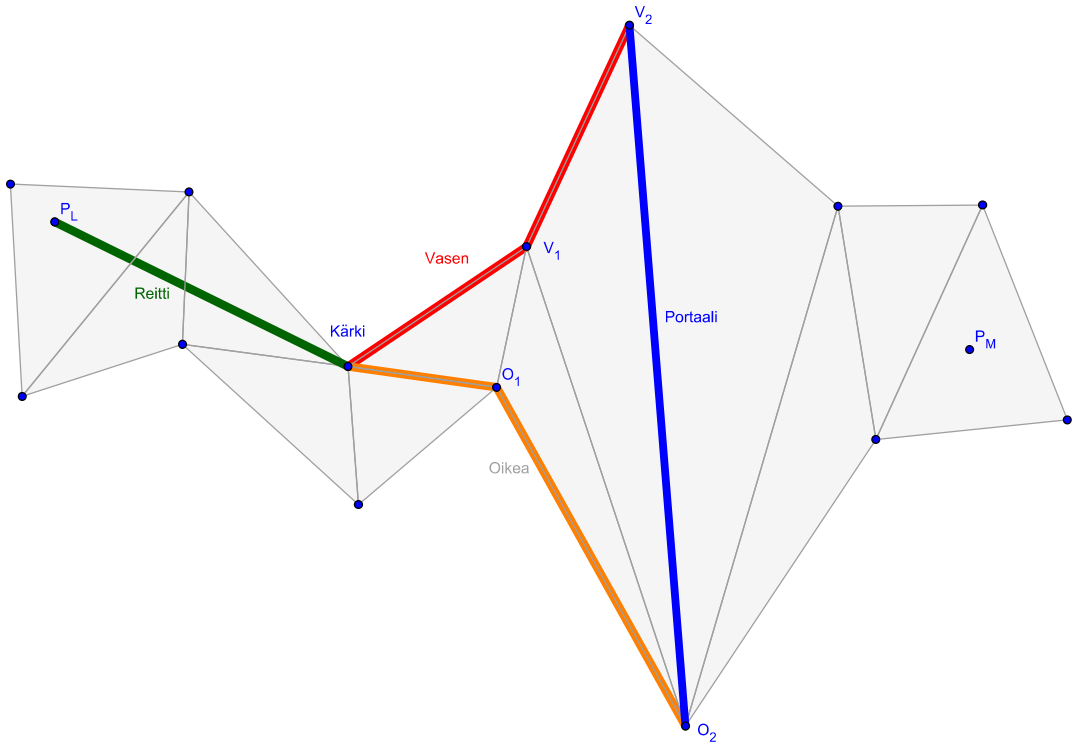
Kuva 5.1: Esimerkki kanavasta.

Kanava ei yksikäsitteisesti määrää pelihahmon varsinaista reittiä. Koska pelihahmon on luvallista liikkua navigaatioverkkomallin alueella, sen on luvallista liikkua missä vain kanavan alueella. Jos oletamme, että lyhyin reitti on edullisin, ja että kanava on muodostettu tämän mukaisesti, niin varsinaisen reitin muodostamiseksi meidän täytyy etsiä suorin reitti, joka kulkee kanavassa. Eräs ratkaisu ongelmaan on suppiloalgoritmi, jonka avulla löydetään lyhyin reitti kanavassa sijaitsevien lähtö- ja maalipisteiden välille [DB06].

5.2 Suppiloalgoritmi

Suppiloalgoritmi löytää lyhyimmän reitin kahden pisteen väille kolmioista muodostuvassa kanavassa lineaarisessa ajassa [LP84]. Algoritmi olettaa, että lähtö- ja maalipisteet sijaitsevat kanavan päätykolmioissa. Lähtö- ja maalipisteet voivat sijaita samassa kolmiossa.

Suppiloalgoritmi ylläpitää neljää eri muuttujaa, jotka ovat reitti, oikea ja vasen seinä sekä kärki. Reitti osoittaa listaan, jossa on pisteitä. Reitti ylläpitää löydettyä lyhyintä reittiä. Reitin sisältämät pisteet ovat etenemisjärjestyksessä, eli kun reitti on valmis, lyhyin reitti kulkee listan vierekkäisiä pisteitä yhdistäviä janoja pitkin. Koska lyhyin reitti kulkee kanavan kolmioita pitkin, reitin sisältämät pisteet ovat maali- ja lähtöpisteet sekä vaihteleva määrä kanavan kolmioiden kulmapisteitä. Seinät ovat myös listoja, jotka sisältävä pisteitä, ja ne on listassa lisäysjärjestyksessä. Seinät edustavat suppilon seiniä, ja niissä ylläpidetään tietoa siitä, missä lyhyimmän reitin seuraava piste täytyy sijaita. Aivan kuten reitin kohdalla myös seinät sisältävät ainoastaan kanavan kolmioiden kulmapisteitä sekä mahdollisesti lähtö- ja maalipisteet. Suppilon kärki osoittaa pisteeseen, johon löydetty lyhyin reitti sillä hetkellä loppuu ja josta suppilon seinät alkavat. Suppilon kärki kuuluu siten aina lyhyimpään reittiin. Kärki on aina molempien seinien ensimmäinen piste. Kuva 5.2 selventää suppiloalgoritmin käyttämiä tietorakenteita. Kuvassa piste P_L ja Kärki muodostavat reitin, pisteet Kärki, V_1 ja V_2 vasemman seinän ja Kärki, O_1 ja O_2 muodostavat vasemman seinän. Pisteiden V_2 ja O_2 välillä on portaali, joka on juuri käsitelty.



Kuva 5.2: Suppiloalgoritmin tila eräässä kanavassa.

Olettakaamme että kanava esitetty listana, joka sisältää portaalit siinä järjestyksessä kuin niiden läpi tulee edetä, jotta lähtöpisteen sisältävästä kolmiosta päästään maalipisteen sisältämään kolmioon. Tällöin suppiloalgoritmi etenee seuraavasti. Aluksi suppilon kärki asetetaan lähtöpisteeseen ja käymme läpi portaalien listaa. Jos portaalien lista on tyhjä, lisätään maalipiste toiseen seinään ja muodostetaan reitti. Muutoin otetaan listasta seuraava portaali ja tarkastellaan portaalia suppilon kärjen näkökulmasta. Portaalin vasemmanpuoleinen piste lisätään suppilon vasempaan seinään, ja portaalin oikeanpuoleinen piste suppilon oikeaan seinään. Jos käsittelyvuorossa on kanavan ensimmäinen portaali, molemmat portaalin pisteet täytyy vastaaviin seiniin. Muutoin toisen pisteet täytyy olla jo vastaavassa seinässä, koska kanava muodostuu kolmioista. Olkoon piste P sen portaalin piste, joka täytyy lisätä vastaavan seinään. Käymme läpi kyseisen seinän pisteitä käänteisessä järjestyksessä. Olkoon P_i seinän käsiteltävä piste ja P_{i-1} seuraava piste. Jos seinässä ei ole kuin yksi piste, tämän pisteen täytyy olla kärki. Tällöin P_{i-1} on vastakkaisen seinän toinen piste. Muodostetaan suora L, joka kulkee

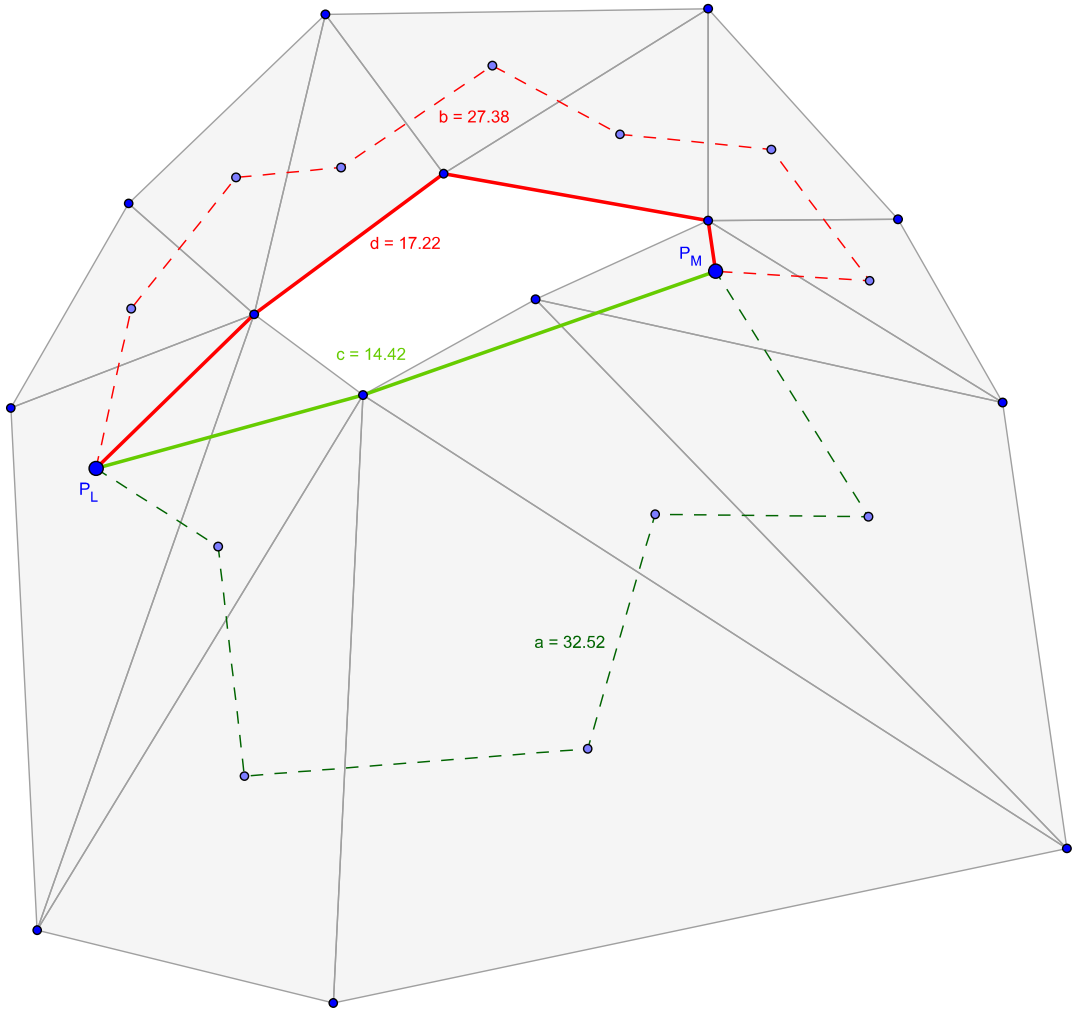
pisteiden P_i ja P_{i-1} kautta. Tarkastellaan pistettä P pisteestä P_{i-1} suuntaan P_i. Jos seinä, johon P:tä lisätään, on oikeanpuoleinen seinä ja P on L:n oikealla puolella, tai jos seinä on vasemmanpuoleinen seinä ja P on L:n vasemmalla puolella, lisätään P seinään ja jatketaan seuraavaan portaaliin. Muutoin seinästä poistetaan piste P_i. Jos P_i on kärki, se lisätään reittiin ja uudeksi kärjeksi määritellään P_{i-1}. Lisätään P_{i-1} käsiteltävään seinään, jolloin se on seinän ainoa piste. Näin jatketaan, kunnes viimeinen portaali on käsitelty. Tällöin maalipiste lisätään toiseen seinään, ja tämän seinän pisteen lisätään sellaisenaan reittiin. Nyt reitti sisältää listan pisteitä, jotka yhdistämällä saamme aikaiseksi lyhyimmän reitin, joka yhdistää lähtö- ja maalipisteitä annetussa kanavassa. Suppiloalgoritmin toimintaa voi kokeilla www-sivulta löytyvällä työkalulla [UB08].

5.3 Kanavan muodostaminen

Kanava voidaan muodostaa helposti seuraavasti. Muodostetaan aluksi navigaatioverkkomallia vastaava reittipisteverkko niin, että reittipisteverkkoon lisätään yksi solmu jokaista navigaatioverkkomallin kolmiota kohden. Solmu voidaan sijoittaa esimerkiksi kolmion painopisteeseen. Tämän jälkeen reittipisteverkon solmujen välille lisätään kaari, jos ja vain jos solmuja vastaavien navigaatioverkkomallin kolmioiden välillä on portaali. Seuraavaksi etsimme ne kolmiot, joissa lähtö- ja maalipisteet sijaitsevat, määritellään näitä kolmioita vastaavat solmun lähtö- ja maalisolmuiksi ja etsimme A*-menetelmällä lyhyimmän reitin näiden solmujen välille. Lopuksi käymme läpi löytyneen reittipisteverkon reitin solmut ja poimimme näitä solmuja vastaavat navigaatioverkkomallin kolmiot. Poimitut kolmio muodostavat kanavan.

Edellä mainittu kanavanmuodostus menetelmä on helppo ymmärtää sekä toteuttaa, mutta siinä on ongelmansa. Ongelma on se, että menetelmä ei välttämättä muodosta sitä kanavaa, jota pitkin navigaationverkkomallin mahdollistama lyhyin reitin kulkisi. Koska etsintä suoritetaan reittipisteverkossa, jonka solmut ovat kolmioiden painopisteissä, jokin reitti saattaa vaikuttaa paremmalta tai huonommalta kuin se todellisuudessa on. Kuva 5.3 havainnollistaa tällaista tilannetta. Kuvassa on kolme murtoviivaa, a, b, c, ja d. Kuvan murtoviivojen viereen on merkitty viivojen pituudet. Murtoviiva c on todellinen lyhyin reitti lähtö- ja maalipisteiden välillä. Murtoviiva a kuvaa reittiä, joka olisi muodostanut kanavan, jota pitkin todellinen lyhyin reitti kulkee.

Murtoviiva b kuvaa reittiä, jonka etsintä totesi lyhyimmäksi, mutta josta muodostuu kanava, jonka kautta muodostettu lyhyin reitti ei ole lyhyin koko navigaatioverkkomallissa.



Kuva 5.3: Ongelmallinen navigaatioverkkomalli.

Ongelmaa on muutamia erilaisia ratkaisuja. Eräs ratkaisu on käyttää muokattua A*-algoritmia nimeltään TA* [DB06]. TA*-algoritmissa reittipisteverkon solmuille luodaan etsinnän edetessä alisolmuja, jotka kuvaavat dynaamisemmin etsinnän tilaa. Menetelmä pyrkii etsimään parempia reittejä jatkamalla parempien reittien etsintää senkin jälkeen,

kun ensimmäinen reitti on löydetty. Toinen vaihtoehto on hylätä kanavan muodostus ja suorittaa lyhyimmän reitin etsintä suoraan navigaatioverkkomallin kolmioiden kulmapisteitä käyttäen.

6 Reitinhaku navigaatioverkkomallissa näkyvyyskartan avulla

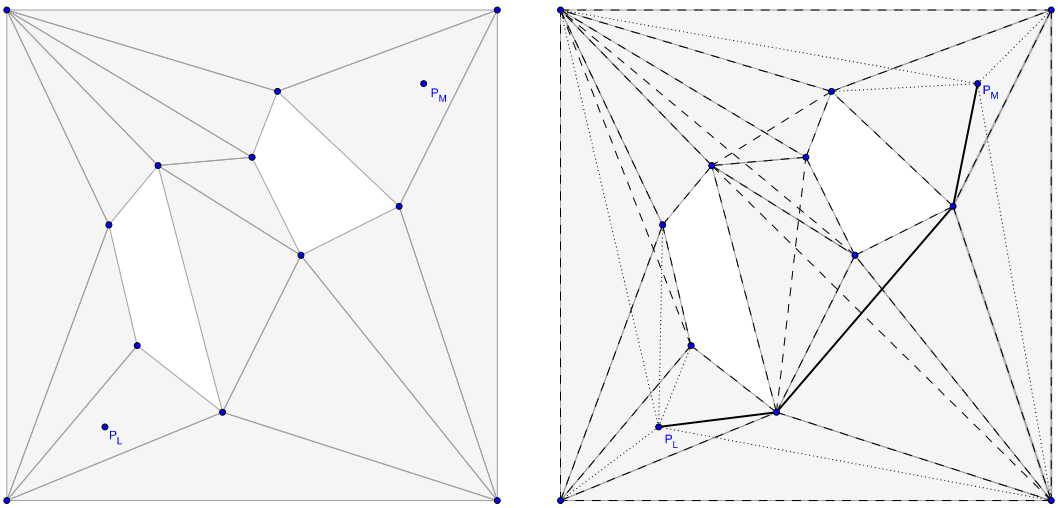
Reitinhaku navigaatioverkkomallissa voidaan suorittaa käyttäen näkyvyyskarttaa [Rab00] [NHT11]. Näkyvyyskartta on yleisesti monikulmioista koostuvaan kuvioon liitetty verkko, josta ilmenee, mitkä sommitelman kulmapisteistä näkevät toisensa. Koska navigaatioverkkomalli koostuu kolmioista, näkyvyyskarttaa voidaan soveltaa myös niihin.

6.1 Näkyvyyskartta navigaatioverkkomallissa

Tämä voidaan toteuttaa käyttämällä näkyvyyskarttaa reittipisteverkon perustana. Näkyvyyskartta muodostetaan kolmioiden kulmapisteiden mukaan niin, että jokaista kulmapistettä vastaa yksi reittipisteverkon solmu. Verkon solmut yhdistettään kaarella vain, jos solmuja vastaavat navigaationverkkomallin kolmioiden pisteet näkevät toisensa. Navigaatioverkkomallissa kaksi pistettä näkevät toisensa vain, jos pisteet voidaan yhdistää janalla niin, että jana ei leikkaa navigaatioverkkomallin reunoja. Luonnollisesti jokaisen kolmion kulmapisteet näkevät toisensa, joten jokaisen kolmion sivuja vastaavat kaaret kuuluvat reittipisteverkkoon. Loppujen kaarien lisääminen naiivilla algoritmilla on triviaalia mutta hidasta. Naiivi algoritmi tutkii, leikkaako jokaisen pisteparin välille muodostettu jana navigaatioverkkomallin reunaa. Jos ei, lisäämme janaa vastaavan kaaren reittipisteverkkoon. Kuva selventää muodostettua reittipisteverkkoa. Muodostettu reittipisteverkko on näkyvyyskartta (eng, visibility graph, vgraph). Näkyvyyskarttoja käytetään esimerkiksi robottien navigoinnissa, ja ne on yleisesti käytetty menetelmä pelihahmojen reitinhaussa. Näkyvyyskartan muodostamiseen on olemassa myös tehokkaampia menetelmiä [GM87].

Jos oletamme, että navigaatioverkkomalli ei muutu pelin aikana, niin näkyvyyskartta voidaan laskea etukäteen ja liittää navigaatioverkkomalliin ennen pelin käynnistämistä. Tällöin näkyvyyskartan muodostamiseen käytetyn algoritmin ei välttämättä tarvitse olla tehokas. Kuitenkin ennen varsinaista reitinhakua lähtö- ja maalipisteitä vastaavat solmut tulee sijoittaa näkyvyyskartasta muodostettuun reittipisteverkkoon. Tämän tapahtuu siten, että muodostetut solmut lisätään reittipisteverkkoon ja tämän jälkeen lisäämme

kaaret näistä solmuista verkon muihin solmuihin vain, jos solmut näkevät toisensa. Lähtö- ja maalisolmut lisätään navigaatioverkkoon eräänlaisina väliaikaisina solmuina, sillä ne joudutaan poistamaan reitinhaun jälkeen, jos edellinen lähtösolmu ei vastaa uutta lähtöpistettä ja edellinen maalisolmu ei vastaa uutta maalipistettä. Kuvassa 6.1 on yksinkertainen navigaatioverkkomalli, jossa P_L on lähtöpiste ja P_M on maalipiste, sekä tästä navigaatioverkkomallista muodostettu näkyvyyskartta. Kuvassa paksu murtoviiva kuvaa lyhyintä reittiä lähtö- ja maalipisteiden välillä, kun näkyvyyskarttaa on käytetty reittipisteverkon tavoin.



Kuva 6.1: Oikealla eräs yksinkertainen navigaatioverkkomalli ja vasemmalla tästä muodostettu näkyvyyskartta. Solmuista P_L ja P_M lähtevät kaaret ovat väliaikaisia. Paksu murtoviiva merkkaa lyhyimmän reitin näiden solmujen välillä.

Kun lähtö- ja maalisolmut on lisätty, reitinhaku suoritetaan esimerkiksi A*-menetelmällä. Jos reitti löytyy, sen täytyy olla optimaalinen. Tämä voidaan perustella seuraavasti. Koska reittipisteverkko oli muodostettu niin, että jokaisesta solmusta on kaari jokaiseen näkyvään solmuun, verkko sisältää kaikki navigaatioverkkomallin mahdollistamat sallitut siirtymät. Koska edelleen navigaatioverkkomallin reunat on edustettuna tässä verkossa, verkko edustaa navigaatioverkkomallin äärireunoja. Joten tästä verkosta löydetyn reitin täytyy olla optimaalinen navigaatioverkkomallin määräämissä rajoissa. Jos lyhyempi reitti olisi olemassa, sen täytyisi kulkea navigaatioverkkomallin reunojen ulkopuolella, jolloin reitti ei olisi kelvollinen.

7 Reitinhaun tehostamismenetelmiä

Dijkstran menetelmä ja sen muunnos A* eivät ole erityisen tehokkaita, jos reittipisteverkko sisältää runsaasti solmuja ja kaaria. Jos reitinhaku on liian raskasta eikä reittipisteverkkoa voida yksinkertaistaa ilman, että pelin toiminta häiriintyisi, ainoaksi mahdollisuudeksi jää tehostaa etsintämenetelmää. On hyvä ymmärtää, että tehostamismenetelmät monimutkaistavat etsintämenetelmiä mahdollisesti tehden niistä vaikeaselkoisia ja hankalasti ylläpidettäviä. Monet menetelmät vaativat myös raskasta etsintäavaruuden esiprosessointia, mikä voi lisää työkaluketjun suorittamisaikaa tai pelin latausaikaa. Tehostamismenetelmien ja esiprosessointien toteuttaminen voi itsessään olla hyvin vaativaa ja aikaa vievää, joten niiden toteuttamisesta saatavia hyötyjä kannattaa harkita perusteellisesti.

Reitinhakua voidaan tehostaa monilla hakuavaruuksien ja Dijkstran ja A*-etsintämenetelmien muunnoksilla [DSS09]. Menetelmät perustuvat jonkin reitinhakua nopeuttavan tiedon lisäämiseen hakuavaruuteen ja etsintämenetelmän muokkaamiseen sellaiseksi, jotta se hyödyntää tätä lisättyä tietoa. Tällaista tietoa ovat esimerkiksi reittipisteverkosta löytyvien hierarkioiden muodostaminen ja verkkoon lisättävät tienviitat sekä näiden yhdistelmät. Monia menetelmiä voidaan edelleen tehostaa esimerkiksi suorittamalla hakua kaksisuuntaisesti.

7.1 Kaksisuuntainen reitinhaku

Dijkstran reitinhakumenetelmästä on olemassa muunnos, jossa haku tehdään yhtäaikaisesti lähtösolmusta maalisolmuun ja maalisolmusta lähtösolmuun. Haut etenevät kunnes molemmat haut löytävät saman solmun, jolloin haku voidaan pysäyttää ja reitti muodostaa käyttäen hyväksi molempien hakujen löytämiä osareittejä [Dan62]. Ajatuksena on, että kun haku tehdään kohteen suuntaan, haut todennäköisesti käsittelevät yhteensä vähemmän solmuja kuin mitä yksisuuntainen haku käsittelisi. Kaksisuuntainen reitinhaku voidaan myös suorittaa rinnakkain [VK11]. Lisäksi se voidaan yhdistää moniin muihin reitinhaun tehostamismenetelmiin ja se on myös monien tehostamismenetelmien olennainen osa [DSS09].

A*-reitinhaku menetelmä voidaan myös suorittaa kaksisuuntaisesti. Koska A*-menetelmä suuntaa etsintää arvioimalla jäljellä olevaa matkaa funktiolla *h(x)*, kaksisuuntainen muunnos tarvitsee kaksi eri funktiota $h_M(x)$ ja $h_L(x)$, joista funktiota $h_M(x)$ käytetään edetessä lähtösolmusta maaliin ja funktiota $h_L(x)$ edetessä. Jos funktiot eivät anna samaa arvio samassa solmussa, reitin optimaalisuudesta ei ole varmuutta kun etsinnät kohtaavat. Funktiot $h_M(x)$ ja $h_L(x)$ voidaan määritellä niin, että ne ottavat huomioon etsinnän molemmat suunnat ja varmistaa kohtaavien reitin optimaalisuuden [Ike94].

7.2 Hierarkioiden hyväksikäyttö

Reittipisteverkoista voidaan löytää erilaisia hierarkioita, joita hyväksi käyttämällä reitinhaku voidaan tehostaa. Reittipisteverkkoon voidaan esimerkiksi määritellä ylemmän tason päällysreittipisteverkkoja, jotka piilottavat alemman tason reittipisteverkon solmut ja toimivat eräänlaisina oikopolkuina [HSW09]. Haku voidaan tällöin suorittaan käyttäen hyväksi näitä oikopolkuja, ja tällöin haku siirtyy käsittelemään alemman tason solmuja vain tarpeen mukaan. Hierarkia muodostetaan jakamalla reittipisteverkko alueisiin, jonka reunoina toimivat solmut. Verkon ylemmällä tasolla nämä reunasolmut yhdistetään kaarilla toisiinsa muodostaen näin ylemmän tason verkon. Verkon tason solmut yhdistetään viereisen tason solmuihin erikoisilla kaarilla, jotka täten mahdollistavat etsinnän etenemisen tasolta toiselle.

Valtatiehierarkioihin (engl. highway hierarcies) perustuvan menetelmä perustuu valtatiekaaren ideaan [SS06]. Kaari on valtatiekaari, jos se kuuluu johonkin lyhyimpään reittiin lähtösolmusta maalisolmuun ja kaari ei sisälly kokonaisuudessaan lähtö- eikä maalisolmun lähiympäristöön. Hierarkian ylempi taso muodostetaan poistamalla kaikki ei-valtatiekaaret sekä ohittamalla kaikki vähiten merkitsevät solmut oikoteillä. Maali- tai lähtösolmun lähiympäristössä haku etenee pitkin alimmin tason verkkoa, joka vastaa alkuperäistä reittipisteverkkoa ja sisältää täten tarkimman tiedon reitin kaarista. Kun haku on tarpeeksi kaukana lähtö- ja maalisolmuista, alemman tason kaarilla ei ole merkitystä perille pääsemisen kannalta. Tällöin haussa voidaan siirtyä ylemmän

hierarkian verkkoon, jossa solmuja ja kaaria on vähemmän. Näin haku voi nopeutua huomattavasti erityisesti silloin, kun alimman tason reittipisteverkko sisältää paljon kaaria ja solmuja.

7.3 Kehittyneemmät suunnatut etsintämenetelmät

A*-menetelmä on suunnattu etsintämenetelmä, koska se suuntaa etsintää kohti maalia välttäen näin epäoleellisten solmujen käsittelyä. On myös muita menetelmiä, joilla etsintää voidaan suunnata ja siten tehostaa. Eräs keino suunnata etsintää on lisätä reittipisteverkkoon eräänlaisia tienviittoja. Nämä tienviitat antavat tietoa siitä, mihin solmuihin solmun tai kaaren kautta pääsee tai mitä kautta johonkin solmuun pääsee helpoiten. Tienviitat muodostetaan reittipisteverkon esiprosessointina ja tämä voi olla aikaa vievää. Toisaalta tienviitat tarvitsevat tallennustilaa, joten ne eivät välttämättä sovellu kaikenlaisiin tilanteisiin, erityisesti silloin jos muistia on rajoitetusti käytössä.

Dijkstran etsintämenetelmän käsittelemiä solmuja voidaan karsia tienviittojen perusteella [WW03]. Reittipisteverkon jokaiseen kaareen tallennetaan tienviitta, joka osoittaa ne solmut, joihin pääsee kaaren kautta kulkevaa lyhyintä reittiä pitkin. Toisin sanoen jokaiseen kaareen k tallennetaan joukko solmuja $L(k)$, joihin kulkeva lyhyin reitti alkaa kaaresta k. Dijkstran etsintämenetelmää muutetaan siten, että käsiteltävä kaari k voidaan ohittaa, jos maalisolmu S_m ei kuulu joukkoon $L(k)$. Muistinkulutusta voidaan vähentää jakamalla solmut ryhmiin geometrisin perustein ja tallentamalla viitat ryhmätasolla sen sijaan, että jokainen solmu tallennettaisiin sinne erikseen. Menetelmän avulla reitinetsinnän aikana käsiteltyjen solmujen määrä voi karsiutua kymmenesosaan.

Tienviittojen käyttöä voidaan edelleen tehostaa yhdistämällä ne esimerkiksi valtatieverkostojen ideaan. Menetelmä nimeltään SHARC osoittaa, että näiden kahden yhdistelmä voi tuottaa tehokkaan etsintämenetelmän, ilman että muistivaatimukset kasvavat kohtuuttomiksi [BD09].

A*-menetelmää voidaan tehostaa menetelmällä nimeltä ALT (engl. A-star, Landmarks and the Triangle inequality) [GoH05]. Menetelmä perustuu maamerkkeihin, jotka ovat erikoisasemassa olevia etsintäavaruuden solmuja. Reitinetsinnän edetessä maamerkkien

välille muodostettuja kolmioepäyhtälöitä tarkkailemalla reittipisteverkon epäedullisilta vaikuttavia haaroja voidaan karsia. ALT voidaan edelleen yhdistää esimerkiksi valtatiehierarkioihin [DSP09].

8 Yhteenveto

Monissa peleissä on pelihahmoja, joilla on erilaisia tavoitteita. Monesti näihin tavoitteisiin pääseminen edellyttää pelihahmo liikkumista nykyisestä olinpaikastaan pelialueen johonkin toiseen paikkaan. Pelin säännöt määräävät liikkumiselle ehdot, ja pelihahmojen tulee noudattaa näitä ehtoja. Lisäksi monissa peleillä on toivottavaa, että pelihahmot liikkuvat luontevasti ja pelihahmon näkökulmasta järkevästi. Jos pelin säännöt määräävät, että pelihahmon tulee liikkuessaan väistää esteet, pelihahmolla täytyy löytää sellainen reitti, jota pitkin liikkumalla pelihahmo voi siirtyä määränpäähänsä esteet kiertäen.

Pelihahmojen liikkumiseen soveltuva alue määrittelee etsintäavaruuden, ja reitinhakumenetelmät etsivät reitit etsintäavaruudesta. Pelien säännöt ja pelialueet yhdessä määrittelevät millainen etsintäavaruus pelille on sopivin. Reittipisteverkko on yleinen etsintäavaruus, koska se soveltuu monenlaisille pelialueille ja koska monet etsintämenetelmät lopulta vaativat reittipisteverkkoa etsintäavaruudekseen. Pelialueelle, joka muodostuu ruuduista, ja jossa pelihahmot sijaitsevat aina kokonaisuudessaan vain yhdessä ruudussa, reittipisteverkko on erinomainen valinta etsintäavaruudeksi. Tällaisia pelejä on esimerkiksi monet lauta- sekä strategiapelit. Monet modernit pelit kuitenkin vaativat hyvin vapaamuotoisia pelialueita, joissa pelihahmojen jokaista sallittua sijaintia ei voida etukäteen määritellä ilman pelin sujuvuuden huonontumista. Tällaisissa peleissä on useasti parempi keskittyä määrittelemään ne alueet, joiden sisällä pelihahmot voivat vapaasti liikkua. Navigaatioverkkomalli on tällainen etsintäavaruus. Siinä liikkumiseen soveltuvat alueet määritellään kolmioilla, jotka voivat sijaita joko kaksi- tai kolmiulotteisessa avaruudessa. Reitinetsintä voidaan tehdä edelleen reittipisteverkossa, joka voidaan muodostaa usealla eri tavalla navigaatioverkkomallista ennen etsintää. Etsinnässä voidaan käyttää esimerkiksi kanavaa tai näkyvyyskarttaa. Kanavaa käyttävässä etsinnässä etsintä suoritetaan kahdessa vaiheessa, joista ensimmäisessä etsitään navigaatioverkkomallin kolmioiden keskipisteiden ja sivujen avulla muodostetusta reittipisteverkosta koostuva kanava, jota pitkin lyhyin reitti todennäköisesti kulkee ja tämän jälkeen lopullinen reitti etsitään kanavasta.

Näkyvyyskarttaa käyttävässä etsinnässä reittipisteverkko muodostetaan ottamalla navigaatioverkkomallin kolmioiden kulmapisteet, sekä etsinnän alku- ja loppupisteet, ja yhdistämällä kaarilla kaikki toisensa näkevät pisteet.

Navigaatioverkkomalli tuo myös oleellista lisätietoa pelialueesta etsintäavaruuteen, jota hyödyntämällä reitinetsinnästä voidaan tehdä joustavampaa pelin sääntöjä rikkomatta. Tätä lisätietoa hyväksi käyttäen pelihahmot voivat esimerkiksi väistää toisiaan ilman reittien uudelleen etsintää. Pelialueelle voidaan tehdä myös kapeita kohtia, joista vain sopivan pienet pelihahmot mahtuvat kulkemaan.

Reitinetsintä voi viedä liikaa aikaa, jos reittipisteverkko sisältävät runsaasti solmuja ja jos pelilaite ei ole riittävän tehokas etsinnän suorittamiseen. Joissakin peleissä reitinetsinnän hidastuminen voi olla niin epätoivottavaa, jolloin reitinetsintää täytyy tehostaa. Triviaali ratkaisu on vähentää reittipisteverkon solmuja, mutta aina tämä ei ole mahdollista. Tällöin ratkaisu voi löytyä erilaisista reitinetsinnän tehostamismenetelmistä, jotka voivat nopeuttaa etsintää huomattavasti. Toisaalta monet tehostamismenetelmät tallentavat lisäinformaatiota reittipisteverkkoon, jolloin sen koko muistissa kasvaa ja etsintämenetelmät monimutkaistuvat.

Pelilaitteita ja pelejä on monenlaisia. Pelilaitteet määrittelevät tarjolla olevat resurssit, joiden puitteissa peliä voidaan suorittaa, ja toisaalta pelien säännöt asettavat vaatimukset, joita pelin ongelmaton suorittaminen vaatii. Jotkin pelilaitteet, kuten uudet pelikonsolit ja PC-tietokoneet, tarjoavat paljon muistia ja runsaasti suoritustehoa, ja toiset, kuten puhelimet ja taskukonsolit, voivat tarjota rajoitetummin muistia ja suoritustehoa. Jos pelissä tarvitaan reitinetsintää, ohjelmoijien tulee punnita eri reitinetsintäavaruuksia ja reitinetsintämenetelmiä pelin vaatimukset ja pelilaitteen resurssit huomioon ottaen ja valita kyseiseen peliin sopivat ratkaisut.

Lähteet

BD09 Bauer, R., Delling, D., SHARC: Fast and robust unidirectional routing. Journal of Experimental Algorithmics (JEA), vol 14, 2009, art. Nro. 4.

Dan62 Dantzig, G. B., Linear Programming and Extensions. Princeton University Press, Princeton (1962).

DB06 Demyen, D. J., Buro, M., Efficient Triangulation-Based Pathfinding. AAAI'06 Proceedings of the 21st national conference on Artificial intelligence. Anthony Cohn, AAAI Press, 2006, sivut 942-947.

Dic03 Dickheiser, M., Inexpensive Precomputed Pathfinding Using a Navigation Set Hierarchy. Teoksessa AI Game Programming Wisdom 2, Charles River Media, 2003, sivut 103-113.

Dij59 Dijkstra, E. W., A Note on Two Problems in Connexion with Graphs. Numerische Mathematik, 1, 1 (1959), sivut 269-271.

DSP09 Delling, D., Sanders, P., Schultes, D., et al., Highway Hierarchies Star. Teoksessa The Shortest Path Problem (Dimacs Series in Discrete Mathematics and Theoretical Computer Science), American Mathematical Society, DIMACS, 2009, sivut 141 – 174.

DSS09 Delling, D., Sanders, P., Schultes, D., Wagner. D., Engineering Route Planning Algorithms. Lecture Notes in Computer Science, vol 5515, 2009, sivut 117-139.

FG97 Franklin, S., Graesser, A., Is it an agent, or just a program?: A taxonomy for autonomous agents, Intelligent Agents III. Agent Theories, Architectures, and Languages, Lecture Notes in Computer Science Volume 1193, 1997, sivut 21-35

GM87 Ghosh, S. K., Mount, D. M., An output sensitive algorithm for computing visibility graphs. SIAM Journal on Computing, 28th Annual Symposium on Foundations of Computer Science, 1987, sivut 11-19

GoH05 Goldberg, A. V., Chris Harrelson, C., Computing the shortest path: A search meets graph theory. SODA '05 Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, 2005, sivut 156-165.

Ham08 Hamm, D., Navigation Mesh Generation: An Empirical Approach. Teoksessa AI Game Programming Wisdom 4, Charles River Media, 2008, sivut 113-123.

HSW09 Holzer, M., , Schulz, F., Wagner, D., Engineering multilevel overlay graphs for shortest-path queries. Journal of Experimental Algorithmics (JEA), vol 13, 2009, art. 5.

Ike94 Ikeda, T., et al. A fast algorithm for finding better routes by AI search techniques. Proc. Vehicle Navigation and Information Systems Conference, 1994, sivut 291 – 296.

LP84 Lee, D. T., Preparata, F. P., Euclidean shortest paths in the presence of rectilinear barriers. Networks, 14, 3(1984), sivut 393-410.

MJ08 McGuire, M., Jenkins, O., Creating Games: Mechanics, Content, and Technology, A K Peters, 2008, sivut 104-105.

NHT11 Nguyet, T. T. N., Hoai, T. V., Thi, N. A., Some Advanced Techniques in Reducing Time for Path Planning Based on Visibility Graph. Third International Conference on Knowledge and Systems Engineering (KSE), 2011, sivut 190-194

Rab00 Rabin, S., A* speed optimizations. Teoksessa Game Programming Gems, Charles River Media, 2000, sivut 272-287.

RN03 Russell, S. J., Norvig, P. Teoksessa Artificial Intelligence: A Modern Approach. Upper Saddle River, N.J.: Prentice Hall, 2003, Sivut 97–104.

Sno00 Snook, G., Simplified 3D Movement and Pathfinding Using Navigation Meshes. Teoksessa Game Programming Gems, Charles River Media, 2000, sivut 288-297.

SS06 Sanders, P., Schultes, D., Engineering Highway Hierarchies. Lecture Notes in Computer Science, vol 4168, 2006, sivut 804-816

Ste02 van der Sterren, W., Tactical Path-Finding with A*. Teoksessa Game Programming Gems 3, Charles River Media, 2002, sivut 294-306.

Stou00 Stout, B., The Basics of A* for Path Planning. Teoksessa Game Programming Gems. Charles River Media, 2000, sivut 254-262.

Toz03 Tozour, P., Search space representations. Teoksessa AI Game Programming Wisdom 2, Charles River Media, 2003, sivut 85-102.

UB08 Universität Bonn, Calculating the shortest path in a simple polygon - the Funnel Algorithm, 25.3.2008 http://web.informatik.uni-bonn.de/I/GeomLab/ShortestPathLP/index.html [21.4.2013]

VK11 Vaira, G., Kurasova, O., Parallel Bidirectional Dijkstra's Shortest Path Algorithm. Proceedings of the 2011 conference on Databases and Information Systems VI: Selected Papers from the Ninth International Baltic Conference, 2011, sivut 422-435.

WC02 White, S., Christensen, C., A Fast Approach to Navigation Meshes. Teoksessa Game Programming Gems 3, Charles River Media, 2002, sivut 307-320.

WW03 Wagner, D., Willhalm, T., Geometric Speed-Up Techniques for Finding Shortest Paths in Large Sparse Graphs. 11th Annual European Symposium, 2003, sivut 776-787.

Yap02 Yap, P., Grid-based Pathfinding. Lecture Notes in Artificial Intelligence, vol 2338 (2002), sivut 44-55.